Python: module cdutil.times

cdutil.times

index

Modules

MA	Numeric	cdtime	sys
cdms.MV	cdms	string	types

Classes

TimeSlicer
    ASeason
        Seasons

class ***ASeason***(TimeSlicer)

Methods defined here:

__init__(self)

Methods inherited from TimeSlicer:

average(self, slab, slices, bounds, norm, criteriaarg=None, statusbar=None)

```
Return the average of the result of slicer
   Input:
      slab                 : the slab on which to operate
      slices               : the slices for each part
      bounds               : the length of each slice
      norm                 : the actual length of each "season
      criteriaarg          : arguments for criteria thing
   Output:
      out : the average of slab, masked by criteria
```

departures(self, slab, slicerarg=None, criteriaarg=None, ref=None, statusbar=None)

get(self, slab, slicerarg=None, criteriaarg=None, statusbar=None)

statusbar1(self, i, n, statusbar)

statusbar2(self, statusbar)

class ***Seasons***(ASeason)

Method resolution order:
Seasons
ASeason
TimeSlicer

Methods defined here:

__init__(self, *seasons)

climatology(self, slab, criteriaarg=None, criteriaargclim=None, statusbar=None)

```
Compute the climatology from a slab
Input:
  slab
  criteriaarg     : the argument for criteria function when s
  criteriaargclim : the argument for criteria function when a
                    if different from criteriarg
Output:
  The Average of the seasons in the order passed when constru
  i.e if DJF and JJA are asked, the output will have the aver
  2 criteria can be passed one for the slicing part and one f
```

departures(self, slab, slicerarg=None, criteriaarg=None, ref=None, statusbar=None)

```
Return the departures for the list of season you specified, r
i.e. if you asked for DJF and JJA and the first season of you
Check your time axis coordinate !!!
To pass a specific array from which to compute departures, pl
for info one default departures see: departures2.__doc__
```

get(self, slab, slicerarg=None, criteriaarg=None, statusbar=None)

```
Get the seasons asked for and return them in chronological or
i.e. if you asked for DJF and JJA and the first season of you
Check your time axis coordinate !!!
slicerarg will be ignored
it is recomended to use Season(slab,criteria=mycriteriaargume
rather than Season(slab,None,None,mycriteriaarguments)
Now for the original doc of the get function see get2__doc__:
```

Methods inherited from TimeSlicer:

average(self, slab, slices, bounds, norm, criteriaarg=None, statusbar=None)

```
Return the average of the result of slicer
   Input:
      slab                : the slab on which to operate
      slices              : the slices for each part
      bounds              : the length of each slice
      norm                : the actual length of each "season
      criteriaarg         : arguments for criteria thing
   Output:
```

```
                out : the average of slab, masked by criteria
```

statusbar1(self, i, n, statusbar)

statusbar2(self, statusbar)

class ***TimeSlicer***

```
    Author : Charles Doutriaux: doutriaux1@llnl.gov
    Date: April 2001
    Returns masked average of specific time slices
    "slicer" determine which slices of the Transient Variable (TV) are
    "criteria" gets TV (with time dimension) and returns a "timeless" ma

    "slicer"
      Input:
            - Time Axis
            - User argument (can be anything) (in a list if more than on
      Output:
          indices             : the indices for each season:
                                                  [[i1,i2,...,il],
                                                  [j1,j2,...,jm],
                                                  ...,
                                                  [k1,k2,...kn]]
          bounds              : the bounds covered by each slice for eac
                                                  [[[i1b1,i1b2],[i2b
                                                  [[[j1b1,j1b2],[j2b
                                                  ...,
                                                  [[k1b1,k1b2],[k2b1
          norm                : the actual length of each "season", and
                                                  [[Li,Si],
                                                  [Lj,Sj],
                                                  ...,
                                                  [Lk,Sk]]
    "criteria"
      Input:
            - slab : a slab
            - mask:  the actual percentage of data in each subset used
                     the bounds of its first (time) dimension must be co
                     they will be used by centroid
            - spread: the begining and end time of the slice processed
            - User argument (can be anything)
      Output:
            - the slab, masked

    Once constructed the object, beside "slicer" and "criteria" has 3 fu

    "get" : which returns the slices wanted, appropriately masked
       Input:
          slab : the slab on which to operate
          sliceruserargument  : anything your slicer function needs, de
```

```
      criteriauserargument: anything your criteria function needs, c
   Output:
      out  : averaged and masked slices of slab

"departures" : which returns the departures of slab from the result
   Input:
      slab                : slab from which the we want to get the c
      sliceruserargument  : anything your slicer function needs, de
      criteriauserargument: anything your criteria function needs, c
      (ref): optional     : result from get or equivalent precompute

   Output:
      out : departure of slab from ref

"average" : which return the average of the result of get
   Input:
      slab                : the slab on which to operate
      slices              : the slices for each part
      bounds              : the length of each slice
      norm                : the actual length of each "season"
      criteriaarg         : arguments for criteria thing
   Output:
      out : the average of slab, masked by criteria

Example of construction:
TS=TimeSlicer(slicerfunc,criteriafunc)
myres=TS(myslab,[[slicerarg,[criteriaarg]])
myresddeparture=TS(myslab,[[slicerarg,[criteriaarg,ref]]]
```

Methods defined here:

__init__(self, slicerfunction=None, criteriafunction=None)

average(self, slab, slices, bounds, norm, criteriaarg=None, statusbar=None)

```
      Return the average of the result of slicer
         Input:
            slab                : the slab on which to operate
            slices              : the slices for each part
            bounds              : the length of each slice
            norm                : the actual length of each "season
            criteriaarg         : arguments for criteria thing
         Output:
            out : the average of slab, masked by criteria
```

departures(self, slab, slicerarg=None, criteriaarg=None, ref=None, statusbar=None)

get(self, slab, slicerarg=None, criteriaarg=None, statusbar=None)

statusbar1(self, i, n, statusbar)

statusbar2(self, statusbar)

Functions

centroid(msk, bounds, coords=None)

```
Computes the centroid of a bunch of point
Authors: Charles Doutriaux/Karl Taylor
Date: April 2001
Input:
  s: a slab
  bounds : the bounds of the overall thing
  coords : the coordinate spanned by each subset
Output:
  centroid: a slab representing the centroid, values are between 0
  centroid is 1D less than s
```

cyclicalcentroid(s, bounds, coords=None)

```
returns the centroid, but this assumes cyclical axis, therefore sp
Usage:
  cyclecentroid=cyclicalcentroid(s,bounds)
Input:
  s: a slab
  bounds : the bounds of the overall thing
  coords : the coordinate spanned by each subset
Output:
  cyclecentroid : slab is same shape than s but without the 1st di
```

dayBasedSlicer(tim, arg=None)

```
slicer function for the TimeSlicer class
select days
Author : Charles Doutriaux, doutriaux1@llnl.gov
Original Date: June, 2003
Last Modified: ...
Input:
  - tim: time axis
  - arg: character string representing the desired day/days or day
         day are represented as "Jan-01" "January-01" "jan-1", "1-
         days can be represented by 2 number but then month is ass
         you can mix definitions
Output:
  -
```

generalCriteria(slab, mask, spread, arg)

```
Default Conditions:
  50% of the data
  AND
  Centroid < x (in absolute value), centroid is always between 0 (
  by default centroid is not used

Author: Charles Doutriaux, doutriaux1@llnl.gov

Usage:
```

```
          generalCriteria(slab,sliced,slices,arg)
          slab : the original slab
          mask:  the actual percentage of data in each subset used to pr
                 the bounds of its first (time) dimension must be correc
                 they will be used by centroid
          spread: the begining and end time of the slice processed
          arg:
              First represent the % of value present to retain a slice
              Second represent the value of the centroid (between 0: per
              If you do not want to use one these criteria pass None
              if you would rather use a cyclicalcnetroid pass: "cyclical
```

getMonthIndex(my_str)

```
      Given a string representing a month or a season (common abrev)
      Returns the ordered indices of the month
      Author: Krishna Achutarao
      Date: April 2001
```

getMonthString(my_list)

```
      Given a list of month creates the string representing the sequence
```

isMonthly(s)

```
      This function test if the data are monthly data from the time axis
```

mergeTime(ds, statusbar=1)

```
      Merge chronologically a bunch of slab
      Version 1.0
      Author: Charles Doutriaux, doutriaux1@llnl.gov
      Usage:
      mymerged=mergeTime(ds)
      where:
      ds is a list or an array of slabs to merge, each slab MUST be in c
      Output:
      a slab merging all the slab of ds
      order is the order of the first slab
```

monthBasedSlicer(tim, arg=None)

```
      slicer function for the TimeSlicer class
      select months
      Author : Charles Doutriaux, doutriaux1@llnl.gov
      Original Date: April 2001
      Last Modified: October, 2001
      Input:
        - tim: time axis
        - arg: character string representing the desired month/season or
               also you can pass a list of the months you want (string c
               you can mix integer and strings
      Output:
        -
```

setAxisTimeBoundsDaily(axis, frequency=1)

```
Sets the bounds correctly for the time axis (beginning to end of d
Usage:
tim=s.getTime()
cdutil.times.setAxisTimeBoundsMonthly(tim,frequency=1)
e.g. for twice-daily data use frequency=2
     for 6 hourly data use frequency=4
     for   hourly data use frequency=24
Origin of day is always midnight
```

setAxisTimeBoundsMonthly(axis, stored=0)

```
Sets the bounds correctly for the time axis (beginning to end of m
Set stored to 1 to indicate that your data are stored at the end c
Usage:
tim=s.getTime()
cdutil.times.setAxisTimeBoundsMonthly(tim,stored=0)
```

setAxisTimeBoundsYearly(axis)

```
Sets the bounds correctly for the time axis (beginning to end of y
Usage:
tim=s.getTime()
cdutil.times.setAxisTimeBoundsYearly(tim)
```

setSlabTimeBoundsDaily(slab, frequency=1)

```
Sets the bounds correctly for the time axis (beginning to end of d
for 'frequency'-daily data
Usage:
cdutil.times.setSlabTimeBoundsDaily(slab,frequency=1)
e.g. for twice-daily data use frequency=2
     for 6 hourly data use frequency=4
     for   hourly data use frequency=24
Origin of day is always midnight
```

setSlabTimeBoundsMonthly(slab, stored=0)

```
Sets the bounds correctly for the time axis for monthly data store
without bounds.
Set stored to 1 to indicate that your data are stored at the end c
Usage:
cdutil.times.setSlabTimeBoundsMonthly(slab,stored=0)
```

setSlabTimeBoundsYearly(slab)

```
Sets the bounds correctly for the time axis for yearly data
Usage:
cdutil.times.setSlabTimeBoundsYearly(slab)
```

setTimeBoundsDaily(obj, frequency=1)

```
Sets the bounds correctly for the time axis (beginning to end of d
for 'frequency'-daily data
Usage:
cdutil.times.setSlabTimeBoundsDaily(slab,frequency=1)
or
cdutil.times.setSlabTimeBoundsDaily(time_axis,frequency=1)
e.g. for twice-daily data use frequency=2
```

```
            for 6 hourly data use frequency=4
            for   hourly data use frequency=24
      Origin of day is always midnight
```

setTimeBoundsMonthly(obj, stored=0)

```
      Sets the bounds correctly for the time axis (beginning to end of m
      Set stored to 1 to indicate that your data are stored at the end c
      Usage:
      tim=s.getTime()
      cdutil.times.setAxisTimeBoundsMonthly(s,stored=0)
      or
      cdutil.times.setAxisTimeBoundsMonthly(tim,stored=0)
```

setTimeBoundsYearly(obj)

```
      Sets the bounds correctly for the time axis for yearly data
      Usage:
      cdutil.times.setSlabTimeBoundsYearly(slab)
      or
      cdutil.times.setSlabTimeBoundsYearly(time_axis)
```

switchCalendars(t1, u1, c1, u2, c2=None)

```
      converts a relative time from one calendar to another, assuming th
      Usage: cvreltime(t1,c1,u2,c2)
      where t1 is cdtime reltime object or a value (then u1 is needed)
      c1,c2 are cdtime calendars
      u1, u2 the units in the final calendar
```

weekday(a, calendar=None)

Data

ANNUALCYCLE = <cdutil.times.Seasons instance>
APR = <cdutil.times.Seasons instance>
AUG = <cdutil.times.Seasons instance>
DEC = <cdutil.times.Seasons instance>
DJF = <cdutil.times.Seasons instance>
FEB = <cdutil.times.Seasons instance>
JAN = <cdutil.times.Seasons instance>
JJA = <cdutil.times.Seasons instance>
JUL = <cdutil.times.Seasons instance>
JUN = <cdutil.times.Seasons instance>
MAM = <cdutil.times.Seasons instance>
MAR = <cdutil.times.Seasons instance>
MAY = <cdutil.times.Seasons instance>
NOV = <cdutil.times.Seasons instance>
OCT = <cdutil.times.Seasons instance>
SEASONALCYCLE = <cdutil.times.Seasons instance>
SEP = <cdutil.times.Seasons instance>
SON = <cdutil.times.Seasons instance>
YEAR = <cdutil.times.Seasons instance>